Chapter 6 - Procedures and Macros

Writing and using procedures

- Avoid writing the same sequence of instruction again and again.
- Write it in a separate subprogram and call that subprogram whenever necessary.
- For that CALL instruction is used.

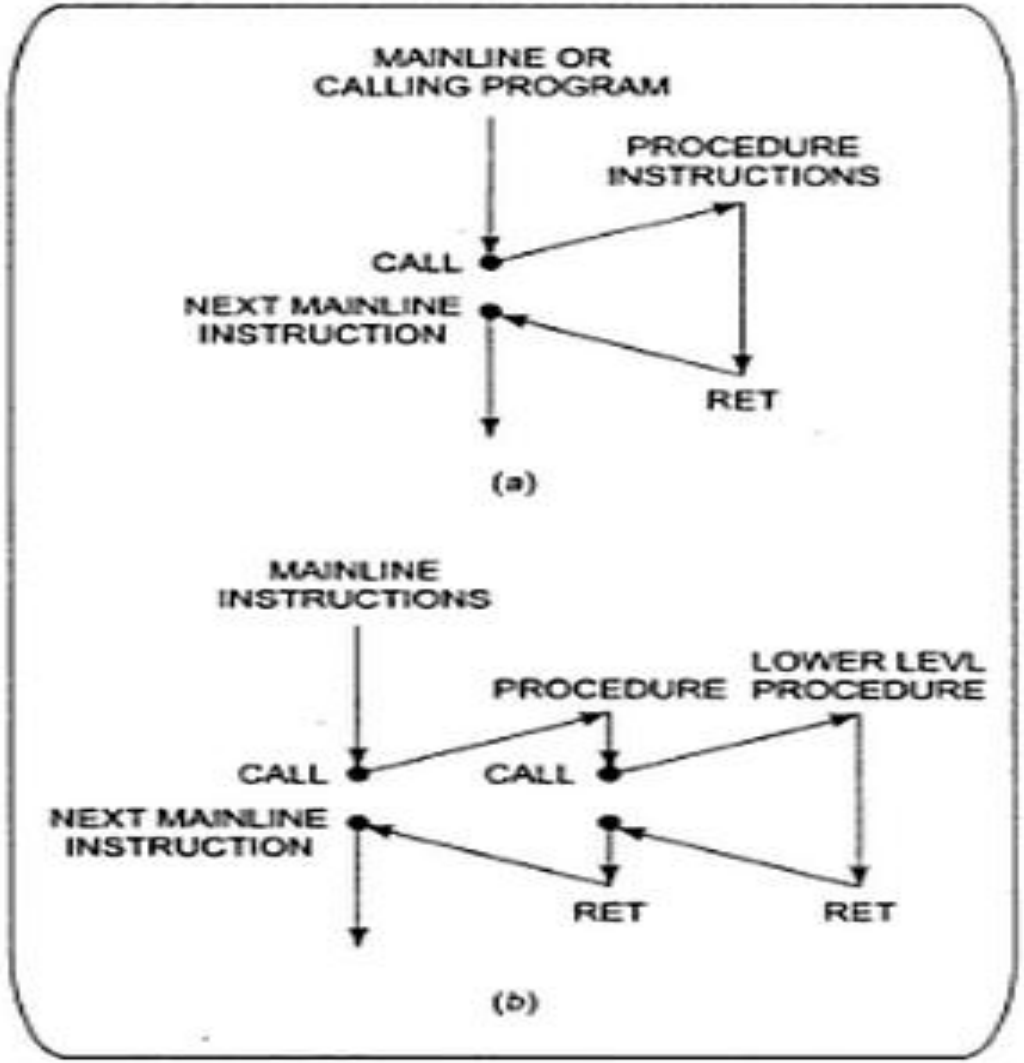
The CALL and RET instructions(contd.)

•Stores the address of the next instruction to be executed after the CALL instruction to stack. This address is called as the return address.

•RET at the end of the procedure, it copies this value from stack back to the instruction pointer (IP).

The CALL and RET instructions(contd.)

Chart for CALL and RET instruction →

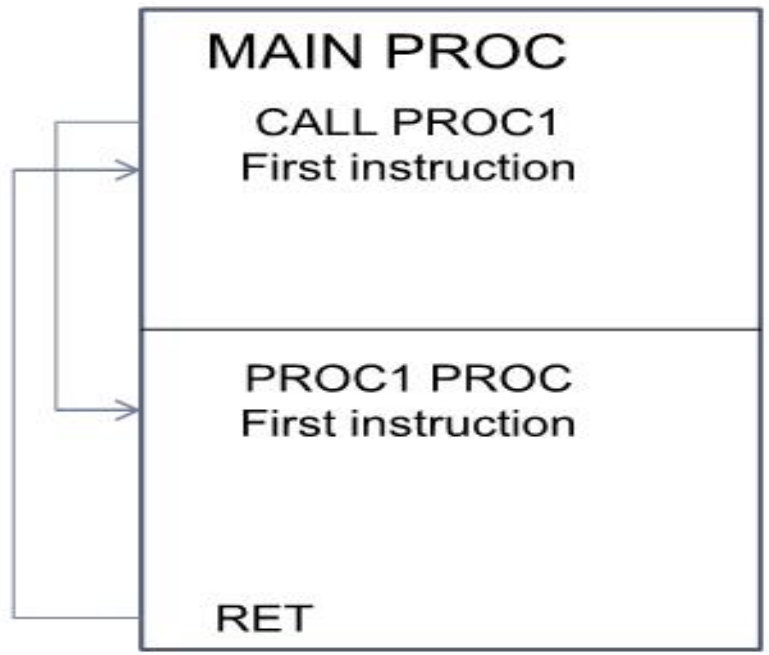


Creating Procedures

- Large problems can be divided into smaller tasks to make them more manageable
- A procedure is the assembly equivalent of a Java or C function
- Following is an assembly language procedure named `sample`:

```
sample PROC
   .
   .
   ret
sample ENDP
```

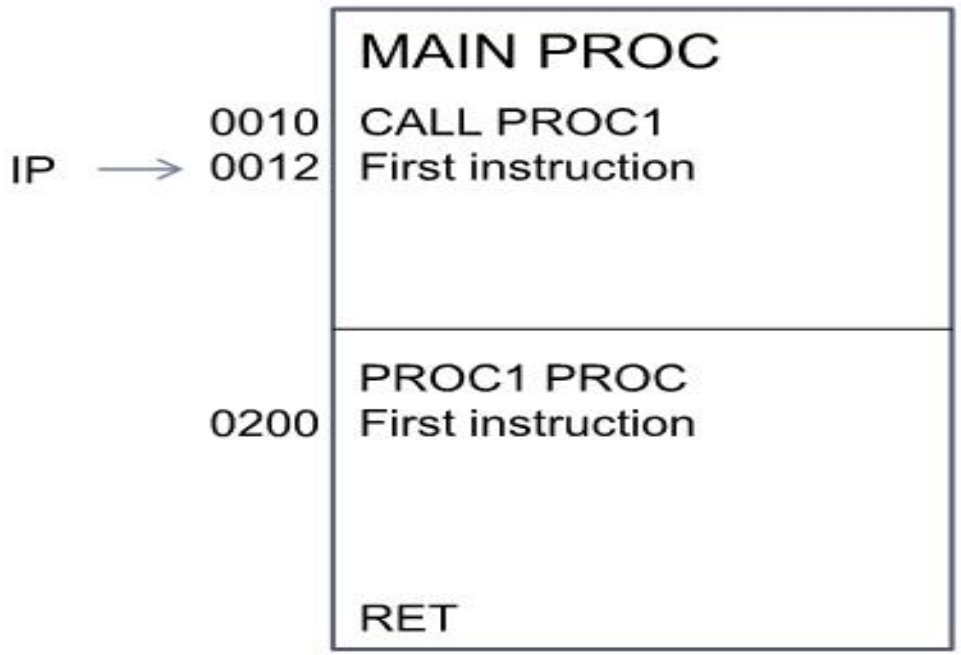
Procedure call and return

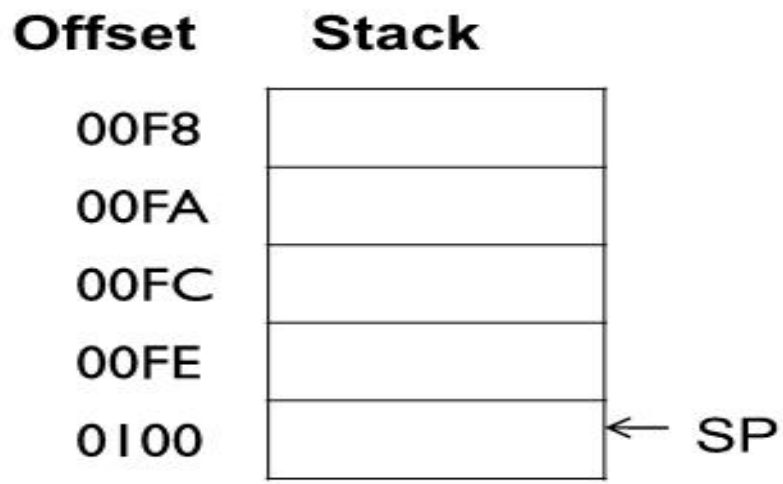


CALL and RET Instructions

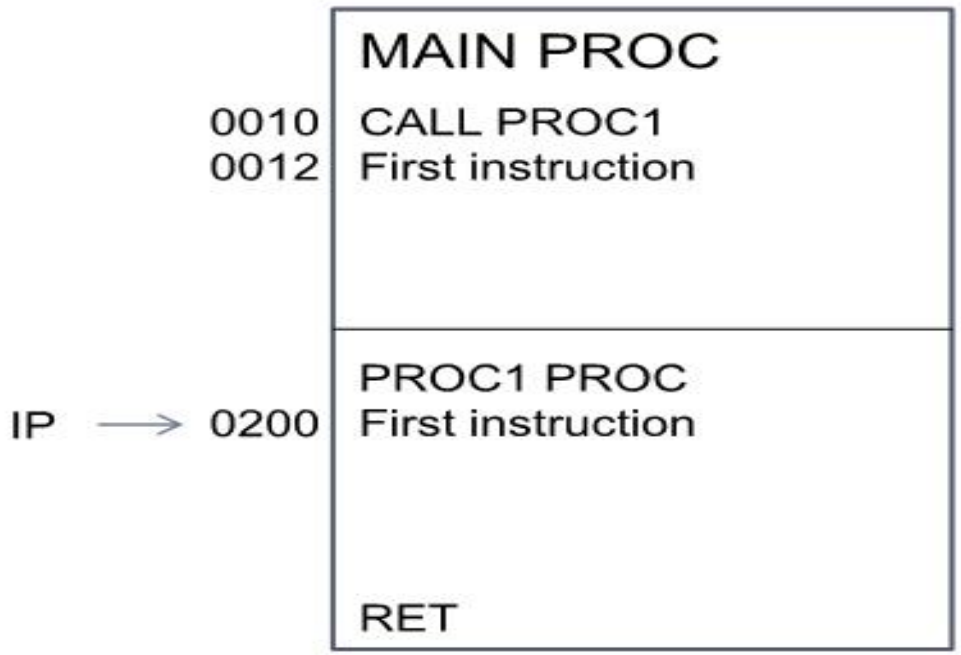
- The CALL instruction calls a procedure
  - pushes offset of next instruction on the stack
  - copies the address of the called procedure into IP (Note: IP=Instruction Pointer)
- The RET instruction returns from a procedure
  - pops top of stack into IP

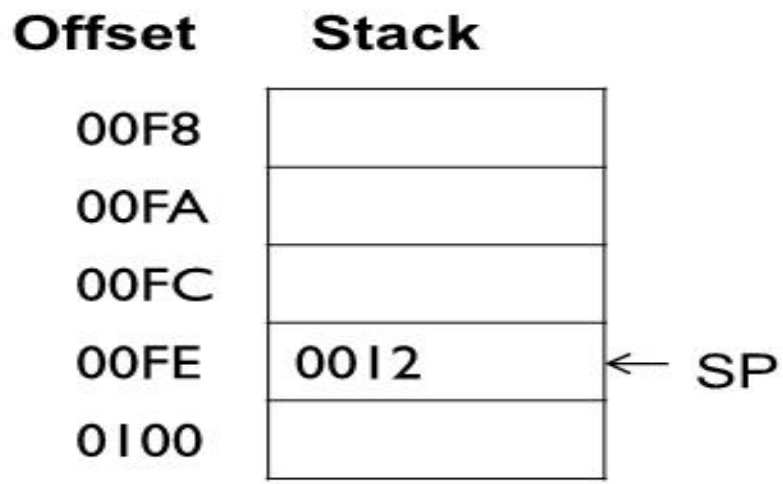
Before Call


MAIN PROC
0010
CALL PROC1
IP
0012
First instruction
PROC1 PROC
0200
First instruction
RET

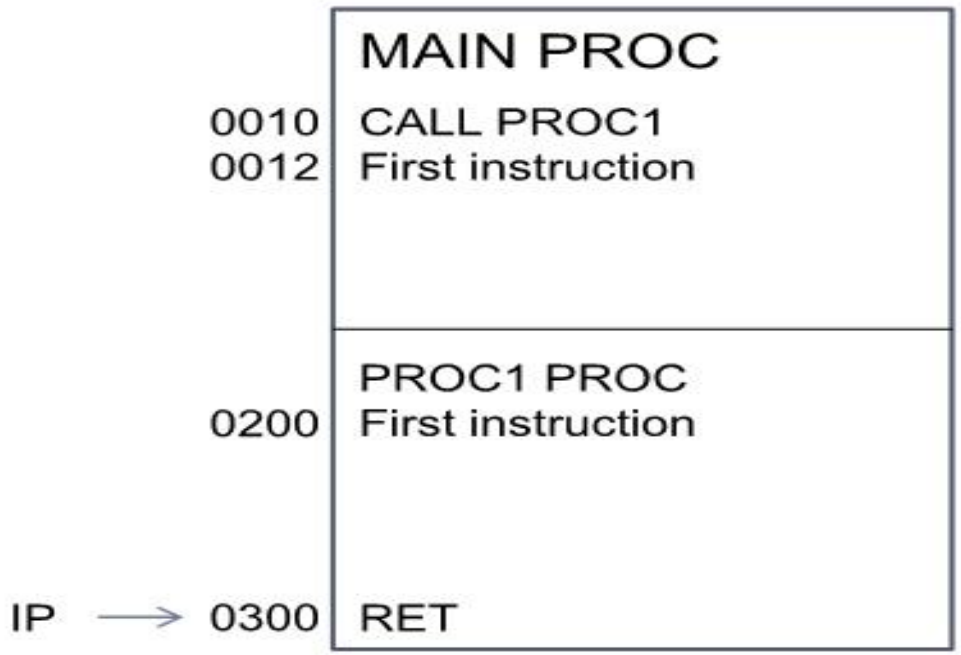

Offset
Stack
00F8
00FA
00FC
00FE
0100
SP

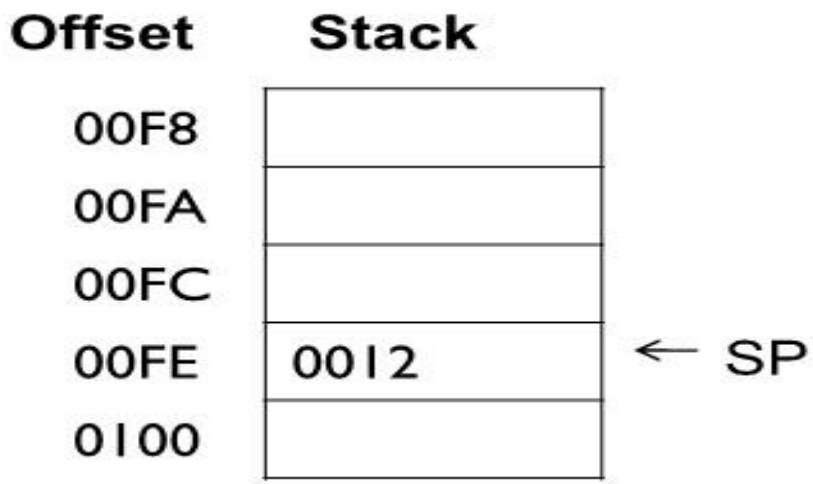
After Call


MAIN PROC
0010 CALL PROC1
0012 First instruction
PROC1 PROC
IP 0200 First instruction
RET

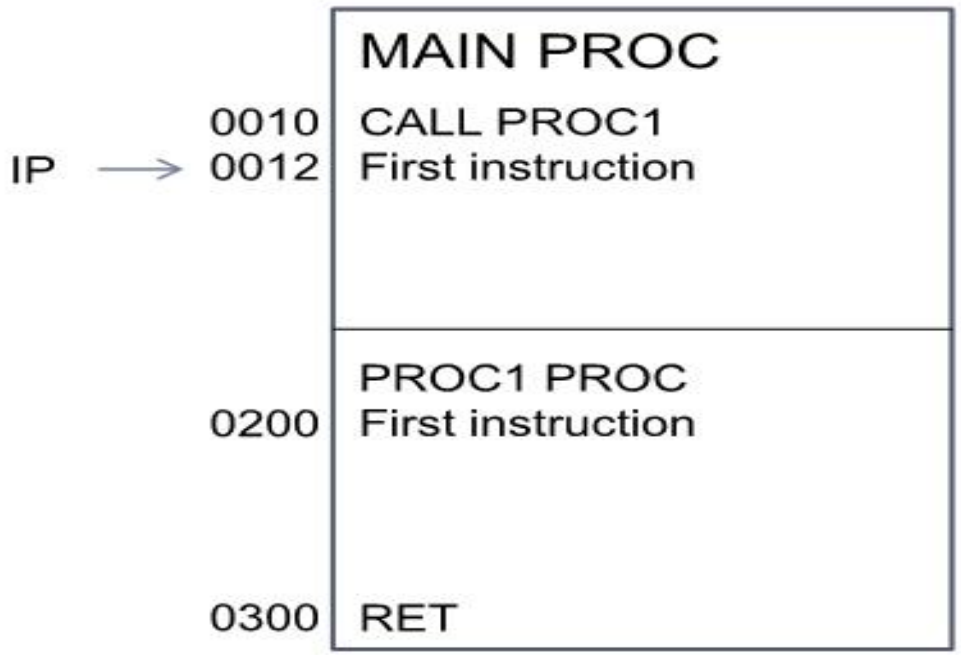

Offset
Stack
00F8
00FA
00FC
00FE
0012
SP
0100

Before RET


MAIN PROC
0010 CALL PROC1
0012 First instruction
PROC1 PROC
0200 First instruction
IP
0300 RET


Offset
Stack
00F8
00FA
00FC
00FE
0012
SP
0100

After RET


MAIN PROC
0010
CALL PROC1
IP
0012
First instruction
PROC1 PROC
0200
First instruction
0300
RET


Offset
Stack
00F8
00FA
00FC
00FE
0100
SP

The CALL and RET instructions(contd.)

Types of CALL :

- WITHIN-SEGMENT **NEAR CALL**: produce the starting address of the procedure by adding a 16-bit signed displacement to the contents of the instruction pointer.

- INTERSEGMENT **FAR CALL**: used when the called procedure is in different segment.

Difference between NEAR and Far

NEAR

- Within Same CS
- Replace old IP with new IP
- Value of IP is Pushed on the stack
- Also called as **Intrasegment** call

FAR

- Within Different CS
- Replace old pair CS:IP with new pair
- Value of pair CS:IP is Pushed on the stack
- Also called as **Intersegment** call

Using PUSH and POP

- The PUSH register/memory instruction decrements the stack pointer by 2 and copies he contents of the specified 16-bit register or memory location to memory at the new top-of-stack location.
- The POP register/memory instruction copies the word on the top-of-stack to the specified 16-bit register or memory location and increments the stack pointer by 2.

Passing parameters to and from procedures

Major ways of passing parameters to and from a procedure:

- In register
- In dedicated memory locations accessed by name
- With the stack

Reentrant and Recursive procedures

- **Reentrant procedures:** The procedure which can be interrupted, used and “reentered” without losing or writing over anything.
- **Recursive procedure:** It is the procedure which call itself.

Writing and using Assembler Macros

Comparison Macros and Procedures

- A big advantage of using procedures is that the machine codes for the group of instruction in the procedures needs to be loaded in to main memory only once.
- Disadvantage using the procedures is the need for the stack.
- A macro is the group of instruction we bracket and give a name to at the start of the program.
- Using macro avoids the overhead time involved in calling and returning from a procedures.

Defining and calling a Macro without parameters

```
PUSH-ALL  MACRO
          PUSHF
          PUSH AX
          PUSH BX
          PUSH CX
          PUSH DX
          PUSH BP
          PUSH SI
          PUSH DI
          PUSH DS
          PUSH ES
          PUSH SS
ENDM
```

Defining and calling a Macro with parameters

Syntax:

NameMacro MACRO [parameter1, parameter2...]

Code of the macro

ENDM